Санкт-Петербургский государственный университет

Кафедра компьютерного моделирования и многопроцессорных систем

Зинатов Рустам Раисович

Выпускная квалификационная работа бакалавра

Сравнительный анализ движков для работы с Big Data

Направление 010300

Фундаментальная информатика и информационные технологии

Научный руководитель,
доктор технических наук,
профессор кафедры
Дегтярев А. Б.

Санкт-Петербург

2017

Содержание

Введение

Термин "Большие данные" возник в 2008 году, что по меркам IT-индустрии уже достаточно давно. С тех пор проблема обработки больших объемов информации не стала менее актуальной. Каждый день человечество воспроизводит колоссальное количество данных, и все они нуждаются в обработке. Для решения этой проблемы были разработаны инструменты для работы с Большими данными. Начиная разработку решения задачи сегодня, нам необходимо сделать выбор в пользу той или иной технологии. Кроме того, если поставленная задача не может быть решена с использованием одного инструмента, то помимо эффективности каждой технологии по-отдельности необходимо учитывать их способность работы в связке.

Помимо обработки, данные также нуждаются в хранении. На сегодняшний день популярным решением являются NoSQL базы данных, которые позволяют хранить слабоструктурированные данные и обладают способностью к горизонтальному масштабированию. Распределенность данных физически означает, что связь между вычислительным узлом кластера и сервером базы данных в рамках организованной сети может состоять из двух и более маршрутизаторов.

В данной исследовательской работе была поставлена следующая цель - выбрать задачу и провести сравнительный анализ работы движка и СУБД в связке, выявить наиболее эффективную пару при условии взаимодействия вычислительного кластера и сервера базы данных по глобальной сети.

Для достижение поставленной цели исследования были определены следующие задачи:

- Рассмотреть существующие технологии для обработки Больших Данных и их хранения.
- Выбрать и реализовать тестовую задачу

- Провести тесты, оценить время работы и эффективность технологий в паре.
- Определить наиболее эффективную пару.

Глава 1. Обзор технологий для обработки и хранения Больших данных

1.1. Big Data

Big Data как термин впервые появляется в статье Клиффорда Линча “Big data: How do your data grow?” для журнала Nature в сентябре 2008 года [1]. В этой статье он поднял проблемы обработки Больших данных, с которой приходится сталкиваться ученым - хранение, обработка, управление, резервное копирование. Также Клиффорд Линч выразил идею о необходимости в новых методах и инструментах, которые упростят работу с Большими данными, тем самым выделяя данную проблему, как отдельное направление в информационных технологиях.

И он не ошибся, данная тема, действительная, сейчас актуальна. В современном мире человечество воспроизводит огромные объемы информации каждый день. Эти данные могут быть научными, например, CERN - Европейская организация по ядерным исследованиям, в наши дни обрабатывает данные с Большого адронного коллайдера по 30 Петабайт ежегодно. Или это могут быть пользовательские данные, которые появляются каждый раз когда мы пользуемся банковской картой, телефоном, планшетом, персональным компьютером, или даже фитнес-браслетом. Например, социальная сеть Facebook каждый день получает по 500 Тбайт новых данных от своих пользователей. И с каждым днем объем данных, которые мы воспроизводим растет. Большие объемы данных возникают в различных сферах нашей жизни - бизнес, торговля, транспорт, медицина, медиа.

Понятию Big Data было дано много определений, и на сегодняшний день под этим термином можно понимать:

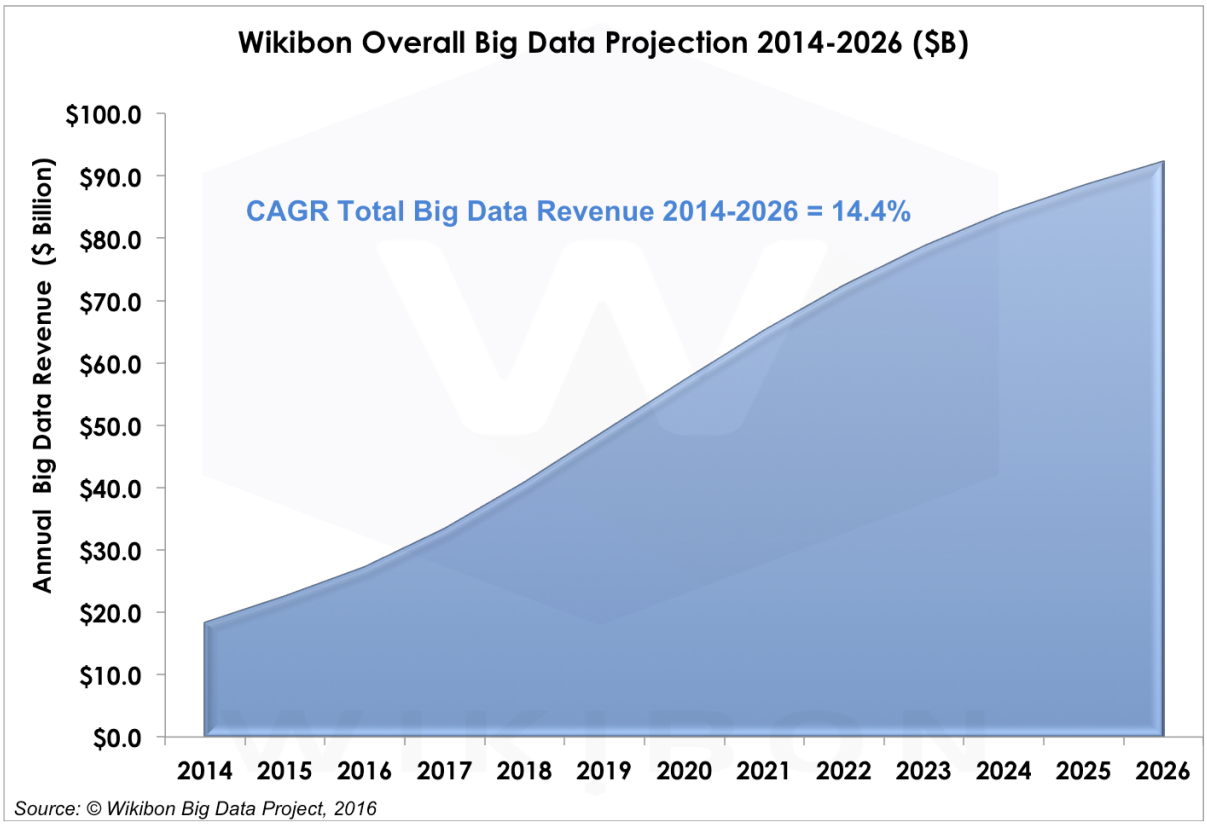
- структурированную и неструктурированную информацию, поступающую в огромных объемах, которую невозможно обработать традиционными инструментами
- совокупность подходов, инструментов и методов обработки информации для получения воспринимаемых человеком результатов, позволяющую проводить распределенные вычисления и масштабироваться в условиях роста входных данных
- все вышеперечисленное

Может возникнуть вопрос - зачем эти данные обрабатывать, какие могут быть приложения Больших данных? Задачей обработки Больших данных является нахождение новых знаний, выявление зависимостей, которые в них сокрыты. Примеров применения может быть много. Например, компании, занимающиеся предоставлением медиа и развлекательных услуг могут использовать данные о предыдущих выборах клиента, а также общие тренды, чтобы угадывать вкусы и предпочтения и делать предложения, в которых он, скорее всего, заинтересован, будь то товары, фильмы и ТВ-шоу, музыка или друзья в социальных сетях.

Также Большие данные нашли свое применение в банковском деле, где с их помощью выявляют случаи мошенничества с ценными бумагами или кредитными картами, осуществляют оценку общего настроения клиентов, используя открытые данные из социальных сетей, чтобы выявить тенденции, оценивают размер резерва для капитала, учитывая риски невозврата кредитов, и многое другое.

Использование Больших данных широко применяется и в других областях. Согласно статье Mike Wheatley для Silicon ANGLE Media, глобальный рынок Больших данных вырастет с 18.3 миллиардов долларов в 2014 году до 92.2 миллиардов долларов к 2026 году, демонстрируя

стабильный ежегодный рост в 14.4 процентов [2], что показывает их актуальность.



Говоря о Больших данных нельзя не упомянуть о так называемых “Three V’s”, которые отражают главные особенности работы с ними.

- Первое “V” - это Volume, объем, одна из главных черт Больших данных, которая не позволяет работать с ними, используя традиционные подходы.
- Второе “V” - Velocity, скорость поступления данных. Важная характеристика, если Вам необходимо обрабатывать данные в реальном времени или близком к нему, и сохранение данных в носители для отложенной обработки не имеет смысла. Вышеупомянутый Facebook должен успевать обрабатывать 900 миллионов изображений в день, причем пользователи ожидают, что фотографии появятся мгновенно, сразу после их публикации.

- Третье “V” - Variety, разнообразие данных. На практике, в реальной жизни, данные могут быть представлены в различных формах - текст, фото, видео, например, с камер наблюдений. Эти данные могут быть структурированными, как логи сервера, или неструктурированными как, например, набор литературных произведений на естественном языке или посты на Twitter. И со всеми этими видами данных необходимо уметь работать.

На сегодняшний день к этим трем характеристикам добавляют еще одно “V” - Veracity, то насколько точны, достоверны данные. Для коммерческой компании важно полагаться на точные данные, чтобы не понести убытки.

Данные четыре особенности определяют необходимость специальных подходов и инструментов для решения задач, связанных с обработкой Больших данных.

1.2. Apache Hadoop

Что такое Apache Hadoop? Согласно информации, взятой с его главной страницы, “Проект “The Apache Hadoop” разрабатывает программное обеспечение с открытым исходным кодом для надежных, масштабируемых, распределенных вычислений.

Библиотека программного обеспечения The Apache Hadoop - это платформа, которая позволяет осуществлять распределенную обработку больших объемов данных на кластерах с использованием простых моделей программирования. Она разрабатывалась как система, которая способна масштабироваться от одного сервера до тысяч машин, каждая из которых предоставляет локальные обработку и хранение. Вместо того, чтобы надеяться на предоставление обеспечения высокой доступности аппаратным обеспечением, библиотека способна обнаруживать и решать проблемы на

уровне приложения, что обеспечивает высокодоступный сервис на кластере из компьютеров, каждый из которых может быть подвержен сбоям.”[3]

Прародителем Hadoop является проект Apache Nutch - система веб-поиска с открытым исходным кодом, работа над которым началась в 2002 году. Реализация проекта появилась достаточно быстро, но не могла масштабироваться на миллиарды веб-страниц. Однако, в 2003 году компания Google опубликовала статью с описанием своей распределенной файловой системы GFS - Google Filesystem. На основе этой статьи разработчики проекта Apache Nutch реализовали подобную систему с открытым исходным кодом NDFS - Nutch Distributed Filesystem, которая решила проблемы с необходимостью хранения очень больших файлов. В 2004 году компанией Google была опубликована еще одна статья, в которой была описана технология MapReduce. [4] К середине 2005 года в проекте Apache Nutch появилась рабочая версия MapReduce, и все алгоритмы проекта были адаптированы под MapReduce и NDFS. Возможности получившейся системы были гораздо шире веб-поиска, поэтому в 2006 году был создан отдельный проект, получивший название Hadoop. Немного позже, Doug Cutting - один из главных разработчиков Apache Nutch был нанят компанией Yahoo! для дальнейшего его развития в инструмент, который способен работать в веб масштабах. Результат появился в 2008 году, тогда компания Yahoo! заявила, что для поиска использует индекс, вычисленный на Hadoop кластере в 10 тысяч ядер. В январе 2008 года Hadoop стал одним из ведущих проектов Apache.

Основными модулями Hadoop являются:

- Hadoop Common - общие утилиты, которая поддерживают другие модули Hadoop
- Hadoop Distributed File System (HDFS™) - потомок NDFS, распределенная файловая система.

- Hadoop YARN - фреймворк для управлениями ресурсами кластера и планированием задач.
- Hadoop MapReduce - система, основанная на YARN, для параллельной обработки наборов данных большого объема.

HDFS - распределенная файловая система, предназначенная для хранения файлов, больших размеров. Объединяя в себе хранилища разных узлов, для пользователя данная файловая система выглядит как единое целое. HDFS является надстройкой над стандартными файловыми системами. Файлы хранятся блоками по 64 или 128 МБайт. Для контроля над всеми данными используются демоны-процессы: Namenode - процесс, который является координатором для всей файловой системы, хранит всю метаинформацию о файле - расположение всех его блоков, и Datanode - процесс, отвечающий за работу файловой системы на некотором конкретном узле. Чтобы избежать потери данных - HDFS реплицирует все блоки. Данная файловая система не подходит для большого числа маленьких файлов - чем больше файлов - тем больше метаинформации и нагрузки на процесс Namenode.

YARN - менеджер ресурсов, на основе которого, можно запускать MapReduce задачи или задачи, реализованные в Spark. Он берет на себя функции управления ресурсами кластера, запуска и контроля работоспособности узлов, распределения их между приложениями, запускаемыми пользователями.

На сегодняшний день помимо основных компонентов существует целая экосистема Hadoop, которая представляет из себя проекты, предоставляющие дополнительные функции на основе базовой функциональности:

- Ambari™ - инструмент для мониторинга и управления, Apache Hadoop кластеров.

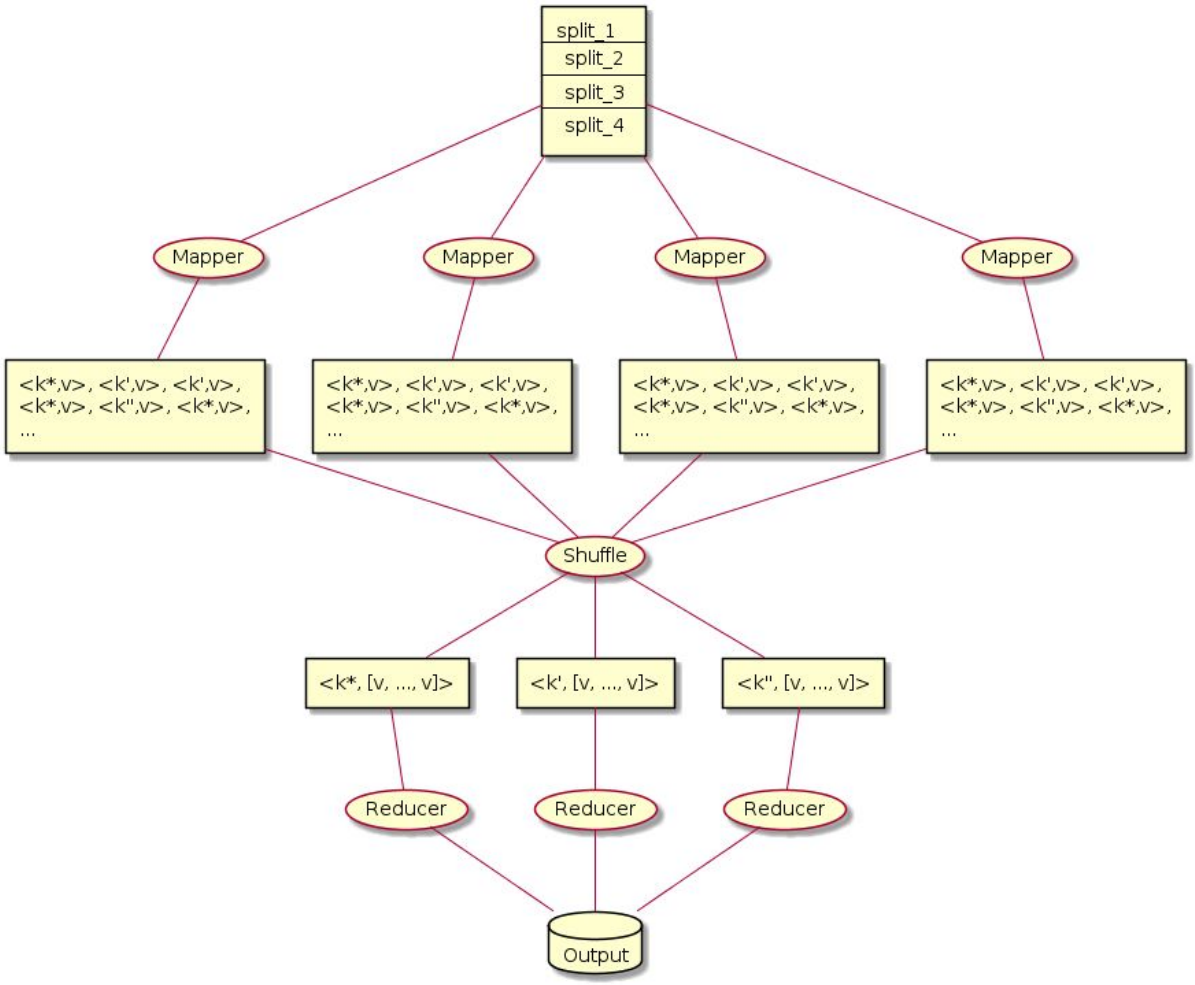
- Avro™ - система для сериализации данных.
- Cassandra™ - масштабируемая СУБД без единой точки отказа.
- Chukwa™ - система сбора данных для управления большими распределенными системами.
- HBase™ - масштабируемая распределенная база данных, поддерживающая структурированное хранение данных для больших таблиц.
- Hive™ - инфраструктура хранилища данных, обеспечивающая обобщение данных и специальные запросы.
- Mahout™ - масштабируемая библиотека для машинного обучения и data mining.
- Pig™ - высокоуровневый язык данных и среда выполнения для параллельных вычислений.
- Spark™ - быстрый движок для обработки Hadoop данных
- Tez™ - фреймворк, основанный на Hadoop YARN, который предоставляет механизм выполнения произвольного дерева (связный ациклический граф) из задач
- ZooKeeper™ - высокопроизводительная служба координации для распределенных приложений.

В этой главе мы подробнее рассмотрим Hadoop MapReduce и Spark.

1.3. Hadoop MapReduce

Hadoop MapReduce - это фреймворк, который предоставляет возможность создания надежных, устойчивых к сбоям приложений, которые обрабатывают огромные объемы данных параллельно на больших кластерах (тысячи узлов). [5]

Данный фреймворк использует модель вычислений MapReduce. Рассмотрим подробнее, как именно она реализуется в рамках фреймворка.



Входные данные задачи могут быть представлены в любом виде. Чтобы использовать модель MapReduce, мы должны представить их в виде пары ключ-значения. Для этого определяется класс, реализующий интерфейс InputFormat, который разделит входные данные на куски (splits), и класс, реализующий интерфейс RecordReader, который поделит каждый split на пары ключ-значение. Hadoop предоставляет несколько стандартных реализаций, например, TextInputFormat - для чтения текстовых данных, FileInputFormat - для чтения каждого файла, как отдельного spit, DBInputFormat - для чтения из реляционной базы данных.

Теперь можно начинать вычисления. Все пары ключ-значение из одного split попадут на один Mapper - процесс, выполняющий функцию map. Эта функция используется для преобразования каждой пары в одну или

несколько новых пар ключ-значение. Чтобы ее определить - необходимо реализовать класса-наследника класса Mapper и переопределить функцию map. Также можно переопределить функции setup и cleanup, которые будут вызваны для каждого Mapper один раз - в начале и в конце вычислений, соответственно.

Промежуточные данные, между фазами Map и Reduce сохраняются на узлах кластера, в локальную файловую систему, все пары будут сгруппированы и отсортированы, для каждого ключа станет известен набор всех его значений и затем эти данные отправятся на фазу Reduce.

Reducer - процесс, который принимает пары - ключ и набор всех значений для этого ключа. Важно, что выполняется правило - все значения, одного ключа попадут на один Reducer. Аналогично классу Mapper, мы определяем свой класс Reducer, реализуя класса-наследника класса Reducer и переопределяя метод reduce.

Для записи результата - используются классы, реализующие интерфейсы OutputFormat и RecordWriter.

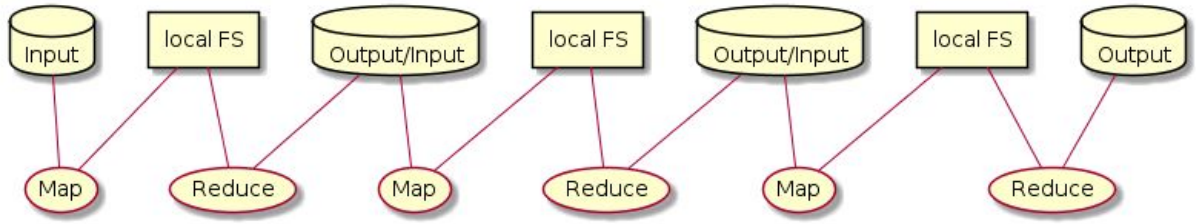
Помимо определения функций map и reduce, возможно определение класса, который будет выполнять предобработку данных между фазами Map и Reduce - класс Combiner. Этот класс, как и Reducer, производит свертку, обрабатывая все значения для некоторого ключа, однако он делает это только для всех пар ключ-значение одного конкретного Mapper. Данная возможность используется для снижения размера данных, которые будут переданы между фазами. Также можно переопределить класс Partitioner, который для каждой пары ключ-значение определит номер процесса Reducer, на который должна отправиться эта пара.

Эффективность данной модели заключается в том, что работа над данными на фазах Map и Reduce ведется параллельно, все пары обрабатываются независимо друг от друга.

Несмотря на это, есть возможность передать на все процессы - Mapper'ы или Reducer'ы общие данные с помощью Distributed cache. Эти данные будут доступны процессам только на чтение. Однако, это снизит эффективность и имеет смысл лишь в случае, если разделяемые данные имеют незначительный размер.

Управление процессами в рамках задачи полностью ложится на менеджер ресурсов - YARN, разработчику остается определить функции map, reduce и то, как читать и записывать данные.

На практике задачи редко укладываются в модель MapReduce, многие из них решаются в несколько MapReduce задач.



Традиционный язык программирования для описания задачи - Java. Также, используя Hadoop Streaming, можно написать программы, реализующие Mapper и Reducer (Combiner и Partitioner опционально), которые принимают данные в стандартный поток ввода - stdin, а результат записывает в стандартный поток вывода stdout на любом языке, которые могут писать в эти потоки.

1.4. Apache Spark

Apache Spark - мощное средство для реализации распределенных вычислений с открытым исходным кодом, которое обеспечивает надежность, высокую скорость работы и простоту использования.[6]

Spark появился благодаря Matei Zaharia в рамках мероприятия AMPLab в Калифорнийском университет в Беркли в 2008 году, и в 2010 году стал проектом с исходным открытым кодом. В 2013 году Spark стал одним из

проектов Apache Software Foundation. В феврале 2014 Spark становится проектом высшего уровня Apache Project. В ноябре 2014, Databricks - компания основателя Spark - M. Zaharia's, устанавливает новый мировой рекорд в сортировке используя Spark. В 2015 году Spark имел свыше 1000 контрибьюторов и становится одним из самых активных проектов Apache Software Foundation.

Важнейшей концепцией Spark является Resilient Distributed Dataset (RDD) - устойчивый распределенный набор данных. RDD - это абстракция, представляющая неизменяемую отказоустойчивую распределенную коллекция объектов, которые могут быть обработаны параллельно. RDD получают из внешних источников, как входные данные, или распределяя уже существующие в программе коллекции.

В отличии от Hadoop MapReduce Apache Spark не ограничивается моделью MapReduce. Существует около 80 операций, которые могут выполняться параллельно и позволяют лаконично описать алгоритм, состоящий из нескольких MapReduce итераций. Над RDD можно выполнять 2 типа операций:

- Transformation - входным аргументом принимает одно или несколько RDD, а результатом является новое RDD.
- Action - завершающая операция, вычисляющая некоторое значение над RDD. После выполнения над RDD action операции RDD закрывается и становится недоступным для использования в других выражениях.

Transformation операции являются ленивыми (lazy) - они не производят вычислений над всем RDD. Вместо этого Spark лишь запоминает последовательность преобразований. Вычисления запускаются, когда над RDD производится action операция. Это позволяет Spark спланировать эффективное выполнение последовательности преобразований. Примерами transformation операций являются:

- map - преобразование каждого элемента RDD
- filter - отсеивает элементы, которые не удовлетворяют некоторому условию
- flatMap - преобразование каждого элемента RDD в некоторое множество элементов
- groupByKey - если элементы представлены в виде пар ключ-значение, то операция сгруппирует все значения для каждого ключа
- reduceByKey - аналог reduce функции в фреймворке MapReduce. Все значения, соответствующие определенному ключу будут агрегированы, и над ними будет проведена коллективная операция
- union - объединяет два RDD
- join - объединяет два RDD, элементы которых представлены в виде ключ-значение. В качестве результата возвращает новое RDD, состоящее из пары - ключ и пара значений из двух исходных RDD

Примерами action операций могут быть:

- count - возвращает число элементов в RDD
- collect - возвращает элементы RDD в виде коллекции
- reduce - вернет результат коллективной операции, проведенной над всеми элементами RDD

Любую программу в Spark можно представить в виде дерева, где листы - входные данные, а корень - результат.

Другим преимуществом является то, что Spark по возможности не записывает промежуточные данные на диск, он использует память для передачи данных между итерациям, что снижает время выполнения задачи.

Как и в Hadoop имеется возможность объявить данные, доступные только для чтения всем узлам кластера с помощью Broadcast переменных.

Также есть возможность определить общие переменные Accumulators, доступные только для изменения - увеличения на некоторое число.

Ядро Spark берет на себя управление памятью, планирование и контроль процессов на кластере, взаимодействие с источниками данных. В случае, если узел выйдет из строя - Spark перезапустит процесс этого узла, так как ему известна вся последовательность операций, однако это негативно скажется на времени выполнения, поэтому фреймворк по возможности сохраняет результаты предыдущего RDD в памяти.

Spark может использоваться совместно с мощными высокоуровневыми библиотеками, которые расширяют его возможности:

- SparkSQL - компонент, позволяющий запрос данных либо при помощи SQL, либо посредством Hive Query Language.
- Spark Streaming - фреймворк может обрабатывать данные в виде потока в реальном или близком к реальному времени.
- MLlib - библиотека для машинного обучения
- GraphX – все они будут подробно рассмотрены в этой статье.

  Spark может работать с HDFS и менеджером ресурсов YARN.

На данный момент существует API для написания программ с помощью JAVA, Python, Scala.

1.5. CouchDB

Одной из рассматриваемых систем управления базами данных была выбрана CouchDB - документо-ориентированная СУБД с открытым исходным кодом, разработанная Apache Foundation и нацеленная на использование в веб-приложениях.

CouchDB хранит базы данных - коллекции из JSON-документов.

Документы – это единицы данных в CouchDB, каждый из которых имеет уникальный идентификатор и состоит из полей и прикреплений (attachments). Поля документа имеют уникальные имена и содержат значения определенных типов (text, number, boolean, list и др.), нет ограничения на размер текста или количество элементов. Описание объектов из реальной жизни с помощью JSON-документов более естественно, чем с помощью реляционных таблиц. Например, рассмотрим визитные карточки. Когда мы их себе представляем, то думаем о имени, номере телефона, адреса почты или другой контактной информации как о чем-то целом - едином документе. Мы не думаем о разрозненных строках в отдельных таблицах.

Увидев визитную карточку, каждый из нас без труда распознает информацию с нее несмотря на то, что визитные карточки могут в той или иной степени отличаться друг от друга. Поэтому в отличии от реляционных СУБД, CouchDB не требует строгой схемы, которой соответствуют все документы, что также делает ее более удобной при описании информации на практике.

Чтение базы данных никогда не блокируется и никогда не надо ждать тех, кто записывает или читает. Любое количество клиентов могут читать документ без блокировки или прерываний из-за одновременного редактирования, даже того же самого документа. Операция чтения CouchDB использует модель управления параллельным доступом с помощью многоверсионности (Multi-Version Concurrency Control MVCC), при которой каждый клиент видит согласованный снимок базы данных в определенный момент времени.

Модель обновления документов CouchDB неблокирующая и оптимистичная. Приложение клиента, загружает документ к себе, применяет изменение, и сохраняет его обратно в базу. Если редактирования на тот же документ, сделанные другим клиентом сохранят изменения раньше, клиент

получит ошибку конфликта редактирования. Для того чтобы разрешить конфликт редактирования, клиенту будет загружена самая последняя версия документа, изменения применятся к этой новой версии и будет проведена повторная попытка записи изменений.

Документы проиндексированы в B-деревьях по специальным порядковом ID (_rev - revision). Каждое редактирование базы данных генерирует новый порядковый номер. Эти индексы редактируются одновременно, с моментом создания или удаления документа. Когда документы CouchDB редактируются, все данные и связанные индексы записываются на диск и транзакционное фиксирование (transactional commit) всегда оставляет базу данных в полностью согласованном состоянии, база данных никогда не содержит частично измененных документов. Кроме того, индексы, реализованные B-деревьями используются для более быстрого поиска в базе данных записанной на диск. [7]

Со временем в базе данных накапливаются неактуальные версии документов. Занятое без пользы пространство очищается по расписанию, или когда в файле базы данных превышен лимит объема ненужных данных. Процесс уплотнения клонирует все активные данные в новый файл и затем удаляет старый файл. База данных остается полностью доступной все время и все обновления и чтения могут быть завершены полностью корректно. Файл старой базы данных удаляется только когда все данные были скопированы и все пользователи перешли на новый файл.

CouchDB предоставляет RESTful HTTP API для чтения и редактирования (add, edit, delete) документов. С помощью http-запросов можно получить конкретный документ по одному из ключей или диапазона ключей. Однако этого, конечно, недостаточно, необходим способ фильтровать, организовывать данные, которые не разложены по таблицам.

В CouchDB для этого есть представления (views). Представления

строятся динамически и не влияют на сами документы, это дает возможность иметь столько разных представлений данных, сколько требуется. Они определены внутри специальным образом сконструированных документов - design documents и могут реплицироваться во все копии базы данных, как обычные документы. Представления определяются с помощью JavaScript функций, описывающих MapReduce задачу.

Функция map - function(doc), будет вызвана для каждого документа в качестве аргумента, преобразует его необходимым образом и отправит на фазу Reduce.

Функция reduce имеет 3 входных параметра:

- keys - массив ключей
- values - массив значений, соответствующих ключам
- rereduce - флаг, означающий повторный запуск reduce функции

CouchDB не запускает reduce сразу от всех пар, вместо этого reduce обрабатывает лишь часть, а затем вызывается повторно от результатов предыдущих запусков, уже с поднятым флаг rereduce. Запуск reduce без флага rereduce производит промежуточную агрегацию.

Для получения результата представления необходимо сделать http-запрос вида “http://<ip>:<port>/<db>/_design/<design_doc>/_view/<view>”. Чтобы получить результат после reduce фазы - надо дополнительно указать параметр reduce=true. Чтобы пары ключ-значение, которые обрабатывает одна функция reduce имели одинаковые ключи - надо указать параметр group=true.

Чтобы сохранить создание представлений быстрым, генератор представлений (view engine) поддерживает индексы его представлений, и обновляет их, чтобы отобразить изменения в базе данных.

Представления не сохраняют свой результат в новую или

существующую базу данных, что лишает возможности использования нескольких MapReduce итераций.

Помимо всего прочего с версии 2.0 CouchDB позволяет настроить распределение данные между разными физическими серверами - sharding. Что позволяет хранить больше данных и повысить надежность базы данных, настроив репликацию - хранение резервной копии на удаленном сервере.

Распределенность системы означает соединение узлов по сети, которая является ненадежным и нестабильным каналом связи. Согласно теореме Брюера (или теорема CAP), распределенную базу данных можно построить с одновременным соблюдением не более двух из трех свойств: согласованность, доступность и устойчивость к разделению. Чтобы разрешить эту проблему CouchDB жертвует согласованностью и допускает в конечном счете согласованность - eventual consistency. Это достигается с помощью incremental replication - процесс, при котором изменения документов периодически копируются между серверами. Что же происходит, если документ успел измениться сразу в двух базах данных? На такой случай CouchDB обладает автоматическим разрешением конфликтов. Когда две версии документа оказываются в конфликте, то версиия-победитель сохраняется, как наиболее недавняя версия. Вторая версия не удаляется, вместо этого CouchDB сохраняет ее как предыдущую версию в истории документа, чтобы оставить к ней доступ.

1.6. MongoDB

Второй рассматриваемой системой управления базами данных была выбрана MongoDB. Как и CouchDB, MongoDB - это документо-ориентированная NoSQL СУБД. MongoDB хранит данные в базах данных, которые состоят из коллекций. Коллекции - это наборы BSON документов - JSON-подобных структур, которые являются структурными

единицами данных для MongoDB. BSON документы не имеют предопределенной схемы, что позволяет менять структуру базы во время работы приложения.

В отличии от CouchDB, MongoDB позволяет производить динамические запросы. Для проведения запросов MongoDB предоставляет командную оболочку - mongo shell. Команды, по большей части, записываются в виде JavaScript выражений. Например, следующая последовательность команд вставляет новый документ в базу данных testdb в коллекцию users:

```
> use testdb
> db.users.insert({firstname:"John", lastname:"Smith", email: "johnsmith@email.org"})
```

Или можно запросить данные передав параметры агрегации:

```
> db.users.find({firstname:"Ben", age:{$gt : 12}})
```

Параметр firstname:”Ben” - является селектором, документы в выборке обязаны иметь заданное значение в поле firstname, а параметр-диапазон $gt означает, что значение поля “age” должно быть не меньше 12. Оболочка поддерживает и другие параметры - операторы над множествами, булевы операторы и другие.

На самом деле при запросе в базу данных MongoDB возвращает не список документов, а специальный объект - курсор. Этот объект возвращает результирующий набор порциями, чтобы не заполнить всю память сразу и повысить эффективность. При обращении к курсору, он сам, незаметно для пользователя запрашивает очередную порцию данных.

Помимо mongo shell, для взаимодействия с программами написанных на разных языках MongoDB предоставляет библиотеки - MongoDB drivers. На данный момент доступны драйверы для C, C++11, C#, Java, Node.js, Perl, PHP, Python, Motor, Ruby, Scala, Go и Erlang. [8] А также коннекторы для

компонентов экосистемы Hadoop: Pig, Spark, MapReduce, Hadoop Streaming, Hive, Flume.

MongoDB также имеет возможность выполнения MapReduce задач на своих базах данных. MapReduce задачи в MongoDB состоят из трех функций: map, reduce, finalize и объекта query - который описывает фильтр.

Map - вызывается для каждого документа, который удовлетворяет условиям query.

Reduce - функция промежуточной агрегации. Когда накапливается несколько пар с одним ключом, MongoDB может выполнить для них функцию reduce. Результатом будет пара ключ-значение, которая потом обрабатывается вместе с остальными, как если бы она была результатом функции map. Это накладывает на функцию ограничения:

Значение, возвращаемое функцией reduce должно быть того же типа, что и значение в паре ключ-значение, возвращаемой функцией map.

Операция reduce должна быть ассоциативной: reduce(key, [A, reduce(key, [B, C])]) == reduce(key, [A, B, C])

Операция reduce должна быть идемпотентной: повторный вызов функции reduce от полученной пары ключ-значение не должен влиять на результат

Порядок в котором функция reduce обрабатывает пары, не должен влиять на результат

Finalize - функция, которая будет вызвана от финальной пары ключ-значение.

В отличии от CouchDB, MongoDB сохраняет результат MapReduce задачи в отдельную коллекцию, что дает возможность запускать задачи, состоящие из нескольких MapReduce итераций.

В MongoDB существуют вторичные индексы, которые реализованы в виде B-дерева. Индексы применяются для оптимизации запросов, позволяя

произвести запрос без полного сканирования коллекции. Проводить запрос можно используя лишь один индекс.

Поддерживаются простые и составные индексы. Примером простого индекса является индекс по _id, с помощью которого можно быстро отыскать документ по его уникальному идентификатору. Составной индекс позволяет оптимизировать запросы по двум или более полям. Оборотная сторона индексов - запись в базу данных повлечет за собой дополнительные расходы по времени на обновление индексов.[9]

Также MongoDB позволяет реплицировать базы данных между узлами. Репликация нужна, чтобы повысить надежность системы, обеспечить доступность и возможность восстановления данных в случае сбоев в сети или одного из узлов. В MongoDB поддерживаются два варианта репликации - набор реплик и master-slave репликация. Во втором случае запись данных производится сначала на главный узел, а потом остальные узлы асинхронно копируют себе новые данные. Первый вариант аналогичен второму с одним отличием - при сбое главного узла, один из подчиненных узлов берет на себя его функции. Также репликация позволяет балансировать нагрузки.

Одной из сильных сторон MongoDB является сегментирование - sharding. Когда база данных разрастется настолько, что индексы или рабочий набор не помещаются в оперативной памяти сервера, MongoDB начнет обращаться к диску. Операция чтения с диска неприемлемо медленная для нормальной работы сервера. В таком случае можно добавить больше оперативной памяти, однако, если приложение и база данных растут, то предел локальной памяти также в скором будет исчерпан. В этой ситуации нам на помощь придет сегментирование - распределение данных по отдельным узлам. MongoDB имеет встроенный механизм, позволяющий автоматически распределять данные и нагрузку между узлами так, что со стороны пользователя распределенная база данных выглядит, как единое

целое. Документы коллекции делятся на порции, согласно сегментному ключу - одному или нескольким полям документа. Далее происходит миграция - порции копируются на свои узлы. Порции также могут реплицироваться на разных узлах. После настройки кластера MongoDB приложения могут обращаться к нему как к обычному серверу MongoDB.

Глава 2. Оценка эффективности на практике

2.1. Тестовая задача

В качестве тестовой задачи была выбрана задача нахождения статистической меры TF-IDF, которая для каждого документа d (document) из заданного корпуса D и для каждого слова t (term) в этом документе определяет важность слова t в документе d. Чтобы рассчитать TF-IDF, необходимо каждой паре документ-слово <d,w> из корпуса D сопоставить значение tf-idf(t,d,D) по формуле:

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)$$

где tf(t,d) - относительная частота слова (term frequency), а idf(t, D) - обратная частота документа (inverse document frequency).

Относительная частота слова в документе вычисляется по формуле

$$\text{tf}(t, d) = \frac{n_t}{\sum_k n_k}$$

где n_x - абсолютная частота слова x. Чем больше число повторений слова - тем важнее оно для документа.

Обратная частота документа - характеристика для каждого уникального слова, встречающегося в корпусе:

$$\text{idf}(t, D) = \log \frac{|D|}{|\{ d_i \in D \mid t \in d_i \}|}$$

В числителе дроби - общее число документов, константа. В знаменателе - число документов, в которых слово t встретилось. Таким образом, чем больше популярность слова, тем меньше значение idf(t, D) и значение tf-df(t, d, D) - важность слова в документе. Таким образом TFIDF ограничивает влияние широкоупотребляемых слов.[10]

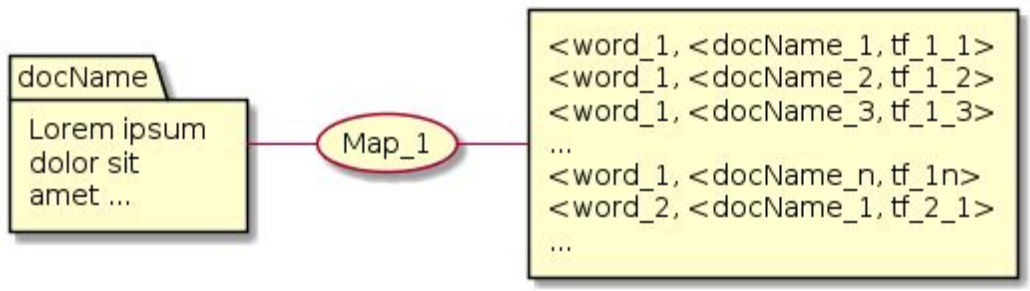
Мера TF-IDF применяется в кластерном анализе. Она представляет документ в виде вектора, значения которого отражают важность некоторого слова корпуса для документа.

Корпус документов может быть большим, а сама задача может быть реализована в модели MapReduce, поэтому она хорошо подходит для проведения тестов.

2.2. Алгоритм решения

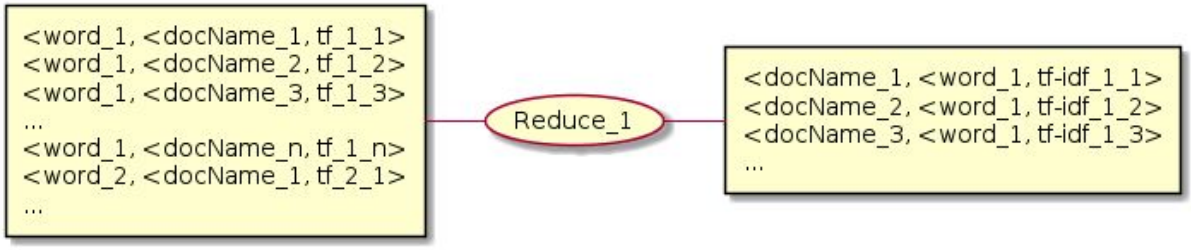
Для решения задачи был использован алгоритм, состоящий из двух MapReduce итераций.

На фазе Map первой итерации на вход поступают пары - название документа и его содержание. Функция map для каждого слова вычислит его относительную частоту в документе и в качестве ключа вернет само слово, а в качестве значения - имя документа и относительную частоту слова в этом документе - tf.



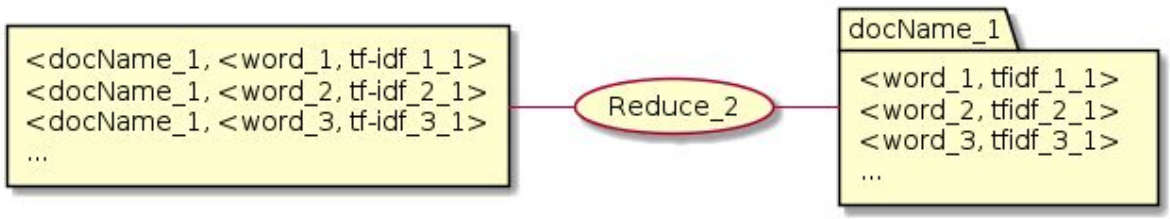
На фазе Reduce первой итерации на вход поступают пары, вычисленные на фазе Map. Все пары для одного слова попадут на один Reducer. Функция reduce соберет все пары, посчитает их количество и получит число документов, в котором встретилось данное слово. Используя найденное значение, можно вычислить обратную частоту документа - idf. Умножив idf на tf в каждой паре мы получим искомый результат. Остается лишь собрать значения tf-idf для документа каждого воедино. В качестве

ключа функция reduce вернет имя документа, а в качестве значения - слово и tf-idf.



Функция map второй итерации ничего не будет делать, просто пробросит данные дальше.

Функция reduce второй итерации на вход получит имя документа, слова и значения tf-idf для каждого слова. В качестве ключа функция reduce вернет имя документа, а в качестве значения - множество пар слово и значение tf-idf.



2.3. Полигон исследования

Для проведения тестов использовался вычислительный кластер из 12 узлов со следующими характеристиками:

- Процессор CPU & Intel Xeon E5440, 2.83Ггц, 8 ядер
- Оперативная память 4Gb
- Жесткий диск HDD ST3250310NS, 7200rpm
- Коммуникация 1Gbit Ethernet

В качестве сервера базы данных использовался ПК с процессором Intel Core i5 и объемом оперативной памяти - 5 GB.

Взаимосвязь вычислительного кластера и сервера базы данных осуществлялась через Интернет.

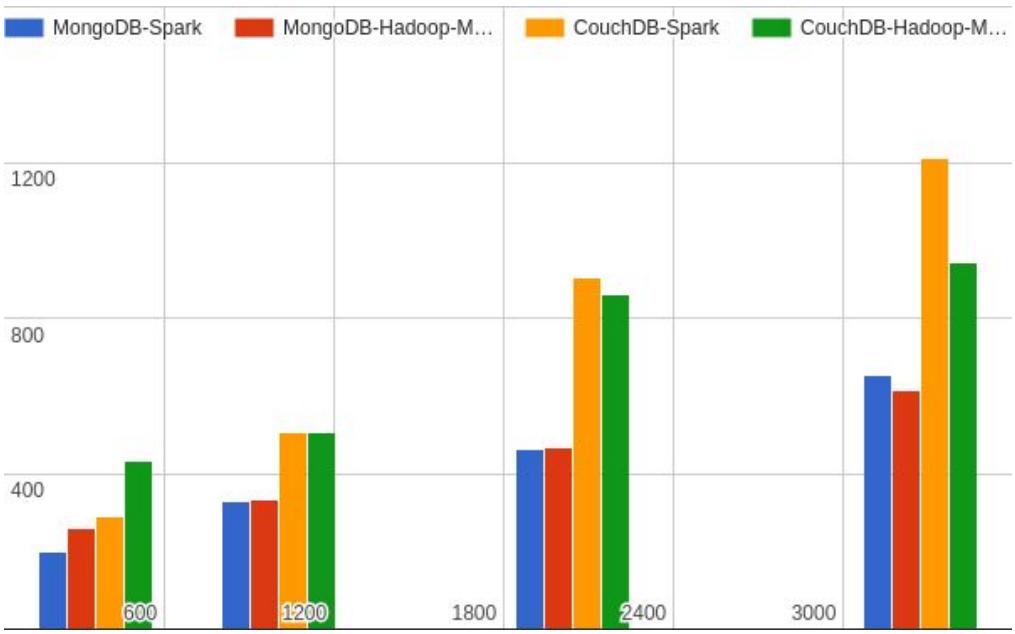
2.4. Проведение тестов

В рамках исследования для решение тестовой задачи было реализовано с помощью разных технологий. Входные данные считывались с базы данных, и записывались тоже в базу данных.

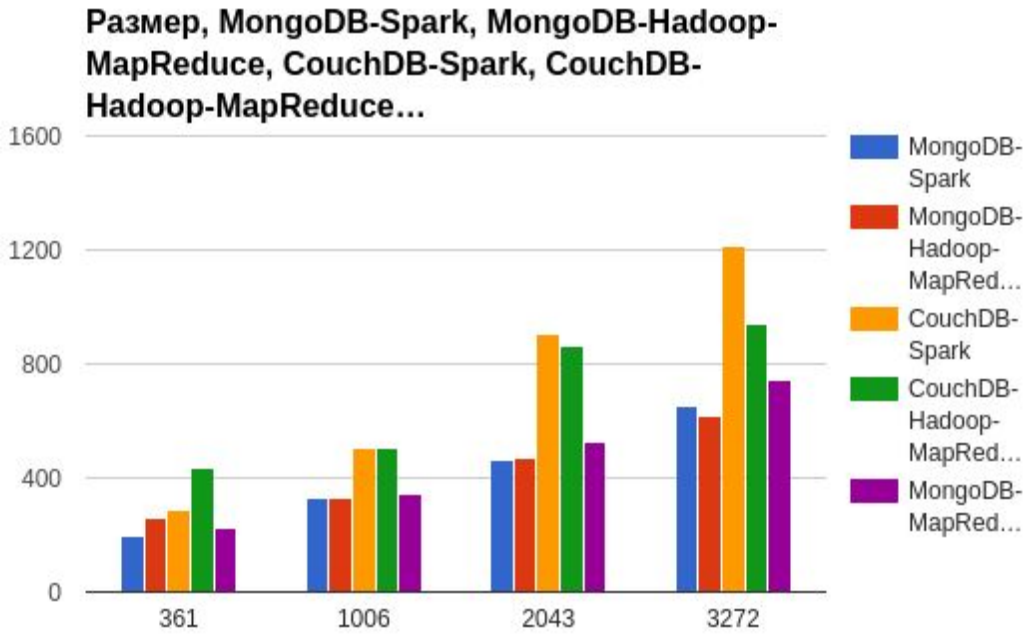
Для работы с CouchDB на основе http-клиента от Apache был реализован connector, обладающий необходимым функционалом - чтение документов из CouchDB блоками заданного размера, запись документов в CouchDB блоками заданного размера.

При работе с MongoDB использовались MongoDB Hadoop и Spark connector-ы, предоставляемые разработчиками MongoDB. Данная разработка позволяет интегрировать данные из базы данных MongoDB и задать параметры этой интеграции.

Ниже приводится результат измерений времени работы.



Можно заметить, что связки, использующие CouchDB проигрывают в скорости работы связкам с MongoDB, т.к. для второй был разработан специальный connector для работы с Hadoop MapReduce и Spark. Рассматривая СУБД MongoDB и CouchDb была затронута тема возможности



запуска MapReduce задач. На рисунке добавлен результат выполнения задачи для MongoDB MapReduce. На небольших объемах данных она не ничем не проигрывает Spark и Hadoop, т.к. ей не приходится скачивать данные по Сети. Поэтому, в случае несложной задачи и небольшого объема данных стоит задуматься о целесообразности развертывания кластера.

Заключение

В рамках работы рассмотрены такие инструменты для работы с Big Data как Hadoop MapReduce, Apache Spark, CouchDB, MongoDB. Реализована тестовая задача для сравнения технологий на практике.

Проведен эксперимент по использованию данных технологий в связке. В результате эксперимента было выяснено, что NoSQL СУБД CouchDB проигрывает в скорости MongoDB при взаимодействии с Hadoop MapReduce и Apache Spark. При работе с удаленной базой данных разницы в эффективности между Hadoop MapReduce и Apache Spark не выявлено, т.к. скорость работы ограничена скоростью Сети.

При небольших объемах данных в рамках одной базы данных MongoDB можно воспользоваться возможностью запускать MapReduce задачи, т.к. это может оказаться быстрее, чем скачивать данные по Сети.

Список цитируемой литературы.

Ссылка на документ в интернете

1. Big data: How do your data grow? https://www.nature.com/nature/journal/v455/n7209/full/455028a.html
2. Wikibon forecasts Big Data market to hit $92.2B by 2026 https://siliconangle.com/blog/2016/03/30/wikibon-forecasts-big-data-market-to-hit-92-2bn-by-2026/
3. What Is Apache Hadoop? http://hadoop.apache.org/
4. MapReduce: Simplified Data Processing on Large Clusters https://research.google.com/archive/mapreduce-osdi04-slides/index.html
5. MapReduce Tutorial https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Purpose
6. Spark Programming Guide http://spark.apache.org/docs/latest/programming-guide.html
7. Кормен Т., Лейзерсон И. Ч., Ривест Р. Л., Штайн К. Алгоритмы: построение и анализ = INTRODUCTION TO ALGORITHMS — 2-е изд. — М.: «Вильямс», 2006. — С. 1296. — ISBN 0-07-013151-1.
8. MongoDB Drivers https://docs.mongodb.com/ecosystem/drivers/
9. Кайл Бэнкер. MongoDB в действии – М.: ДМК Пресс, 2012. – 395 с
10. What does tf-idf mean? http://www.tfidf.com/